Read Stroustrup Chapter 4.

Every name (identifier) in a C++ program has a type ... determines what operations can be applied to the name (that is, to the entity referred to by the name) and how such operations are interpreted. (p.70)

(slides of PH. p.299: bit patterns & MIPS types)

Some of the aspects of C++’s fundamental types, such as the size of an int, are implementation defined. (p.74)

See C.2 and 22.2
(slides of Stru.p.75, p.87)
Problems:

- Memories (late 20th century) are byte addressable, to make programming easier.
- Network messages are communicated, and files are copied into memory byte-by-byte\(^a\), as \textit{streams}.
- Hardware and programming language operations often apply to fixed size data (ints, floats, …) \textbf{larger than} one byte.

(Processing 32-64 bits at a time is more efficient than processing 8 at a time.)

Endianess (byte order) \textbf{only} occur when multibyte data units are treated as separately addressable bytes.

\(^a\) Byt es are called octets by network people

---

RULE 1: Memory is Byte Addressable.

Model of memory: Array of bytes. Each byte is 8 bits.

Here is BIG ENDIAN,
also known as Most Significant Byte byte ordering,
also known as Left to Right ordering

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & \ldots & \ldots & 28 & 29 & 2a & 2b & \ldots & \leftarrow \text{address} \\
\end{array}
\]

\-------------------------------

\begin{array}{cc}
1 & 9 & 9 & 5 & \leftarrow \text{contents} \\
\end{array}

Address is 0x28

1 is the \textit{HIGH} digit
RULE 2: A MULTI-BYTE unit is ALWAYS addressed by the LOWEST NUMBERED BYTE ADDRESS of its bytes.

Here is LITTLE ENDIAN, also known as Least Significant Byte byte ordering also known as Right to Left ordering . . . 2b 2a 29 28 . . . . 5 4 3 2 1 0 <- address

1 9 9 5 <- contents
Address is 0x28
5 is the LOW digit

RULE 3: Hardware processes data in multibyte units correctly.

(X & 0xFF) == low byte (ALWAYS)
((X >> 24) & 0xFF) == high byte (ALWAYS)

lw  $s1, 0x28($0)
    #$s1 = 0x01090905 on Big Endian Machine
    #$s1 = 0x01090905 on Little Endian Machine

andi $s1, $s1, 0xFF
    #$s1 = 0x05  on EVERY MACHINE

lbu $s1, 0x28($0)
    #$s1 = 0x01  on Big Endian Machine
    #$s1 = 0x05  on Little Endian Machine
.data
STR: .asciiz "HEL0"
.text
__start:la $s6,STR
1bu $s0, 0($s6)  #$s0 = 0x48 = 'H'
1bu $s1, 1($s6)  #$s1 = 0x45 = 'E'
1bu $s2, 2($s6)  #$s2 = 0x4C = 'L'
1bu $s3, 3($s6)  #$s3 = 0x4E = 'O'

lw $s4, 0($s6)
# Big Endian Machine(SPARC): $s4 = 0x48454C4E
# Little Endian Machine(PC): $s4 = 0x4E4C4548

The Internet (network order) on Earth is **Big Endian**.
Host orders vary. See htonl, htons, ntohl, ntohs in
/\usr/include/sys/byteorder.h

---

C/C++ Array and Structure Rules
(C character strings ARE arrays)

- **Within one array**, the element sequence
  with increasing subscripts is stored in memory locations
  with increasing addresses.

- **Within one struct (or class)**, the data members
  in order of declaration are stored in memory locations
  with increasing addresses. (Padding is added for alignment.)

- (Successive **bitfields** are stored according to the particular
  computer’s Endianess (byte order). Moral: Don’t use bitfields
  to analyze data in a portable way.)